Lecture 11

Networking
Game Networking Issues

Consistency

• Do our games agree?
  • Where do I see objects?
  • Where do you see them?
  • Who is authoritative?

• How to force agreement?
  • Do I wait for everyone?
  • Do I guess and fix errors?

Security

• What cheats are possible?
  • View hidden data
  • Enter invalid states
  • Improve player skill

• How do we cheat proof?
  • Technical solutions?
  • Community policing?
Game Networking Issues

Consistency

- Do our games agree?
  - Where do I see objects?
  - Where do you see them?
- How to force agreement?
  - Do I wait for everyone?
  - Do I guess and fix errors?

Today’s Lecture

Security

- What cheats are possible?
  - View hidden data
  - Enter invalid
- How do we cheat proof?
  - Technical solutions?
  - Community policing?

Not going to cover
The Issue of Consistency

- **Latency** is root of all evil
- **Local** actions are instant
- **Network** actions are slow

**Example**: targeting
- Want “geometric fidelity”
- Fire a weapon along ray
- Hits first object on ray
- But movement is fast!

How to tell these cases apart?
World State vs. Local State

- **State**: all objects in game
  - **Local State**: on a machine
  - **World State**: “true” state

- **Where** is the world state?
  - On a single machine?
  - Union of local states?

- States may be *inconsistent*
  - Local disagrees with world
  - Is this really a problem?
  - What can we do about it?
The Question of Authority

Centralized Authority

- One computer is authority
  - Stores the full world state
  - Local states must match it
- Often call this the “server”

Distributed Authority

- Authority is divided up
  - Each object has an owner
  - Must match if not owner
- Classically call this “P2P”
Authority and Latency

- Lack of authority enforces a delay
  - Only draw what authority tells you
  - Requires round trip from your input
  - Round-trip time (RTT) can be > 200 ms

- This makes the game less responsive
  - Need some way to compensate for this
Authority and Latency

- Lack of authority enforces a delay
  - Only draw what authority tells you
  - Requires round trip from your input
  - Round-trip time (RTT) can be > 200 ms
  - This makes the game less responsive
  - Need some way to compensate for this

Need to understand basics before solving this
Networking Breaks into Two Phases

### Matchmaking
- Service to find other players
  - Groups players in a session
  - But does not run session
- Why make your own?
  - Control user accounts
  - Implement skill ladders
- 3<sup>rd</sup> party services common
  - Apple GameCenter
  - GooglePlay API
  - CUGL Docker Service

### Game Session
- Service to run the core game
  - Synchronizes player state
  - Supports minor adds/drops
- Why make your own?
  - Must tailor to your game
  - You often have no choice
- Limited 3<sup>rd</sup> party services
  - Often just a networking API
  - For limited class of games
  - **Examples**: Unity, Unreal
Networking Breaks into Two Phases

**Matchmaking**
- Service to find other players
- Groups players in a session
- But does not run session
- Why make your own?
  - Control user accounts
  - Implement skill ladders
- 3rd party services common
  - Apple GameCenter
  - Google OpenMatch
  - CUGL Docker Service

**Game Session**
- Service to run the core game
- Synchronizes player state
- Supports minor adds/drops
- Why make your own?
  - Must tailor to your game
  - You often have no choice
- Limited 3rd party services
  - Often just a networking API
  - For limited class of games
  - **Examples**: Unity, Unreal

Simplify if possible

Our main focus
CUGL Matchmaking

- Requires a custom server
  - Needs a fixed IP address
  - IP is coded into the game
  - Or at least put in an asset
- Can leverage **cloud tech**
  - Write a Docker container
  - Deploy only as needed
- **Benefit**: cross-platform play
  - Must for iOS-Android play
  - See also Open Match
CUGL Matchmaking

Ask for Session

Client

Host

Matchmaker
CUGL Matchmaking

Respond with Room

Client

Host

Matchmaker
CUGL Matchmaking

Ask to Join Room

Client

Host

Matchmaker
CUGL Matchmaking

Connect to Host

Matchmaker

Game Session
CUGL Matchmaking

Reconnect if Dropped

Game Session

Networking
Matchmaking in *Family Style*

**Kitchen Created!**

*Share your kitchen and continue once everyone has joined!*

*Next*
Why Not Just Direct IPs?

- **Idea:** Just let the host be “the server”
  - Player starts up server instance
  - Player writes down their IP address
  - Everyone else types in that IP address

- **Problem:** Network Address Translation
  - Most networks use NAT to attach many devices
  - This means IP addresses on NAT are not real

- Matchmaker provides **NAT punchthrough**!
  - Reason why you keep it open for reconnects
Game Session: Part of Core Loop

Client
- Update
- Draw

Authority
- Update

Networking
Decoupling the Network Loop

Client:
- Local Update
- Draw

Authority:
- Network Update
- Update

Networking
Decoupling the Network Loop

Client

- Local Update
- Draw

Smooth local animation

Authority

- Network Update

Possibly slower tick rate (10 fps)

Should match the client rate

Networking
Decoupling Enables Latency Masking

- Animation is “buying time”
  - Looks fast and responsive
  - But no real change to state
  - Animation done at update

- **Examples:**
  - Players wait for elevator
  - Teleportation takes time
  - Many hits needed per kill
  - Bullets have flying time
  - Inertia limits movement
Game Session: Dedicated Server

- Server developer provides
  - Acts as central authority
  - May be several servers
  - May use cloud services

- **Pros:**
  - Could be real computer
  - More power/responsiveness
  - No player has advantage

- **Cons:**
  - Lag if players not nearby
  - Expensive to maintain
**Game Session: AdHoc Server**

- One client acts as host
  - Acts as central authority
  - Chosen by matchmaker
  - But may change in session

- **Pros:**
  - Cheap long-term solution
  - Can group clients spatially

- **Cons:**
  - Server is a mobile device
  - Host often has advantages
  - Must migrate if host is lost
Game Session: AdHoc Server

- One client acts as host
  - Acts as central authority
  - Chosen by matchmaker
  - But may change in session

- Pros:
  - Predominant commercial architecture

- Cons:
  - Server is a mobile device
  - Host often has advantages
  - Must migrate if host is lost
**Game Session: AdHoc Server**

- One client acts as host
  - Acts as central authority
  - Chosen by matchmaker
  - But may change in session

**Pros:**
- Looks like the CUGL approach?

**Cons:**
- Server is a mobile device
- Host often has advantages
- Must migrate if host is lost
- Authority is distributed
  - Each client owns part of state
  - Special algorithms for conflict
  - Coordinator for adds/drops

- **Pros:**
  - No lag on owned objects
  - Lag limited to “attacks”
  - Same advantages as adhoc

- **Cons:**
  - Incredibly hard to implement
  - High networking bandwidth
Game Session: True P2P

- Authority is distributed
  - Each client owns part of state
  - Special algorithms for conflict
  - Coordinator for adds/drops

- Pros:
  - *Almost* no-one does this outside academia

- Cons:
  - Incredibly hard to implement
  - High networking bandwidth
Game Session: True P2P
Game Session: True P2P

Melee is easy to latency mask!
What Do CUGL Games Use?

- There is a designated host in CUGL networking
  - But this used for matchmaking, not the session
  - No requirement that host is authoritative

- Library was actually designed for P2P
  - Method `send()` broadcasts to all (including host)
  - Worked because *Family Style* spaces were disjoint

- But possible to make host authoritative
  - Method `sendOnlyToHost()` talks only to host
  - Host synchronizes incoming messages
  - Broadcasts back to clients with `send()`
Synchronization Algorithms

- Clients must be **synchronized**
  - Ensure they have same state
  - … or differences do not matter
- Synchronization ≠ authority
  - Authority determines true state
  - Not *how* clients updated
  - Or *when* clients are updated
- Major concept in networking
  - Lots of complicated algorithms
  - Also a **patent mindfield**
  - Take distributed systems course
# Synchronization Algorithms

<table>
<thead>
<tr>
<th>Pessimistic</th>
<th>Optimistic</th>
</tr>
</thead>
</table>
| • Everyone sees same world  
  • Ensure local = world state  
  • Forces a drawing delay  
  • Best on fast networks  
  • Local LAN play  
  • Bluetooth proximity  
  • Or games with limited input  
  • Real time strategy  
  • Simulation games | • Allow some world drift  
  • Best guess + roll back  
  • Fix mistakes if needed  
  • Works on any network  
  • Lag errors can be fixed  
  • But fixes may be distracting  
  • Works great for shooters  
  • Player controls only avatar  
  • All else approximated |
Synchronization Algorithms

**Pessimistic**

- Everyone sees same world
  - Ensure local = world state
  - Forces a drawing delay

- Best on fast networks
  - Local LAN play
  - Bluetooth proximity

- Or games with limited input
  - Real time strategy
  - Simulation games

**Optimistic**

- Allow some world drift
  - Best guess + roll back
  - Fix mistakes if needed

- Works on any network
  - Lag errors can be fixed
  - But fixes may be distracting

- Works great for shooters
  - All else approximated

Also great for distributed authority
**Pessimistic:** Lock-Step Synchronization

- **Algorithm:** play by “turns”
  - Players send turn actions
  - Even if no action was taken
  - Wait for response to render

- **Problems**
  - *Long* Internet latency
  - Variable latencies (jitter)
  - Speed set by slowest player
  - What if moves are lost?

- More common in LAN days
**Algorithm:** turns w/ timeout
- Often timeout after 200 ms
- But can be adapted to RTT
- All moves are buffered
- Executed at end of *next* turn

**Problems**
- Variable latencies (> a turn)
- Speed set by slowest player
- What if moves are lost?

**Used in classic RTS games**

---

**Pessimistic: Bucket Synchronization**

- Player 1
- Player 2
- Synch & Render
- Multiple Moves Possible
- X loss
- time

---

*Networking*
Pessimistic: Bucket Synchronization

- **Algorithm**: turns w/ timeout
  - Often timeout after 200 ms
  - But can be adapted to RTT
  - All moves are buffered
  - Executed at end of next turn

- **Problems**
  - Variable latencies (> a turn)
  - Speed set by slowest player
  - What if moves are lost?

- Used in classic RTS games
Optimistic: Personal State

Local Update

Draw

Network Update

Server Update

Current Network State

Approx. Current State

Unconfirmed player actions

Previous Network State

Current Network State

State updates

Player action confirmations

True State

Networking
Optimistic: Opponent State

- Local Update
- Draw
- Network Update
- Server Update

Current Network State

Approx. Current State

Simulate assuming no actions

Previous Network State

Current Network State

Networking

State updates

Opponent actions

True State
Advantages of Sending Actions

Dead Reckoning

- Assume velocity constant
  - Simulate the new position
  - Treats like physics object
- Generalize to other actions

Error Smoothing

- Can interpolate late actions
  - Create simulation for action
  - Avg into original simulation
- Continue until converge

Networking
The Perils of Error Correction

Networking
CUGL Networking Guarantees

- CUGL built on **Slikenet**
  - Uses **reliable UDP**, not TCP
  - Uses **messages**, not stream
  - Messages are a **byte vector**

- Guarantees **message order**
  - Guarantees are **per client**
  - No guarantee between clients

- Host can **synchronize**
  - Host broadcasts moves to all
  - All clients see in **same order**
CUGL Networking Guarantees

- **CUGL built on Slikenet**
  - Uses **reliable UDP**, not TCP
  - Uses **messages**, not stream
  - Messages are a **byte vector**

- **Guarantees message order**
  - Guarantees are **per client**
  - No guarantee between clients

- **Host can synchronize**
  - Host broadcasts moves to all
  - All clients see in **same order**
CUGL Networking Guarantees

- CUGL built on Slikenet
  - Uses **reliable UDP**, not TCP
  - Uses **messages**, not stream
  - Messages are a **byte vector**

- Guarantees **message order**
  - Guarantees are **per client**
  - No guarantee between clients

- Host can **synchronize**
  - But introduces **message delay**

[Diagram showing network communication and host synchronization]
**Physics: Challenge of Synchronization**

- Deterministic bi-simulation is very hard
  - Physics engines have randomness (not Box2D)
  - Not all architectures treat floats the same

- Need to mix interpolation with snapshots
  - Like error correction in optimistic concern
  - Run simulation forward from snapshots
Physics: Challenge of Synchronization

- Deterministic bi-simulation is very hard
  - Physics engines have randomness (not Box2D)
  - Not all architectures treat floats the same
- Need to mix interpolation with snapshots
  - Like error correction in optimistic concern
  - Run simulation forward from snapshots
Physics: Challenge of Authority

- Distributed authority is very difficult
  - Authority naturally maps to player actions
  - Physics is a set of interactions

- Who owns an uncontrolled physics object?
  - **Gaffer**: The client that set in motion
  - Collisions act as a form of “authority tag”
Summary

- **Consistency**: local state agrees with world state
  - Caused by latency; takes time for action to be sent
  - Requires complex solutions since must draw now!

- **Authority** is how we measure world state
  - Almost all games use a centralized authority
  - Distributed authority is beyond scope of this class

- **Synchronization** is how we ensure consistency
  - Pessimistic synchronization adds a sizeable input delay
  - Optimistic synchronization requires a lot of overhead